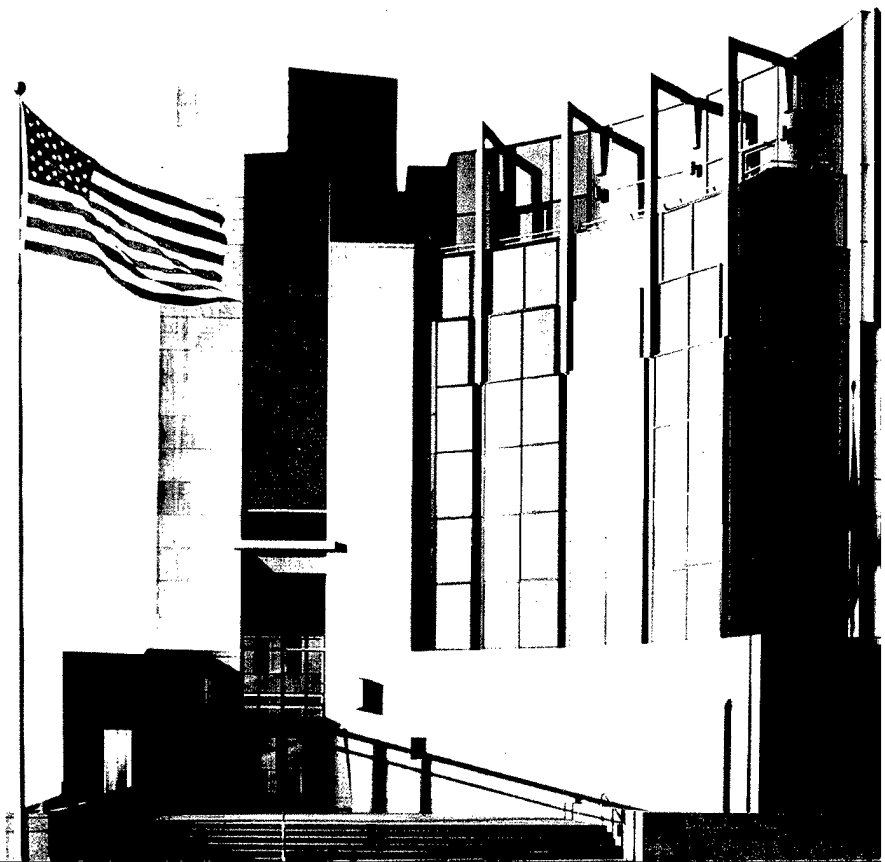
Carnegie Mellon
Software Engineering Institute

The Personal Software Process[SM] (PSP[SM])

Watts S. Humphrey

November 2000



TECHNICAL REPORT
CMU/SEI-2000-TR-022
ESC-TR-2000-022

Carnegie Mellon University does not discriminate and Carnegie Mellon University is required not to discriminate in admission, employment, or administration of its programs or activities on the basis of race, color, national origin, sex or handicap in violation of Title VI of the Civil Rights Act of 1964, Title IX of the Educational Amendments of 1972 and Section 504 of the Rehabilitation Act of 1973 or other federal, state, or local laws or executive orders.

In addition, Carnegie Mellon University does not discriminate in admission, employment or administration of its programs on the basis of religion, creed, ancestry, belief, age, veteran status, sexual orientation or in violation of federal, state, or local laws or executive orders. However, in the judgment of the Carnegie Mellon Human Relations Commission, the Department of Defense policy of "Don't ask, don't tell, don't pursue" excludes openly gay, lesbian and bisexual students from receiving ROTC scholarships or serving in the military. Nevertheless, all ROTC classes at Carnegie Mellon University are available to all students.

Inquiries concerning application of these statements should be directed to the Provost, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-6684 or the Vice President for Enrollment, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, telephone (412) 268-2056.

Obtain general information about Carnegie Mellon University by calling (412) 268-2000.

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

The Personal Software Process℠ (PSP℠)

CMU/SEI-2000-TR-022
ESC-TR-2000-022

Watts S. Humphrey

November 2000

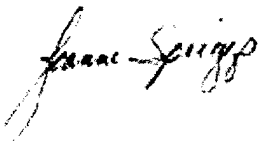
Team Software Process Initiative

This report was prepared for the

SEI Joint Program Office
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER



Joanne E. Spriggs
Contracting Office Representative

This work is sponsored by the U.S. Department of Defense. The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-00-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Acknowledgements

Contributing reviewers for this report were Noopur Davis, Frank Gmeindl, Marsha Pomeroy Huff, Alan Koch, Don McAndrews, Jim McHale, Julia Mullaney, Jim Over, and Bill Peterson.

Abstract

The Personal Software Process[SM] (PSP[SM]) provides engineers with a disciplined personal framework for doing software work. The PSP process consists of a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work. It is introduced with a textbook and a course that are designed for both industrial and academic use. The PSP is designed for use with any programming language or design methodology and it can be used for most aspects of software work, including writing requirements, running tests, defining processes, and repairing defects. When engineers use the PSP, the recommended process goal is to produce zero-defect products on schedule and within planned costs. When used with the Team Software Process[SM] (TSP[SM]), the PSP has been effective in helping engineers achieve these objectives.

This report describes in detail what the PSP is and how it works. Starting with a brief discussion of the relationship of the PSP to general quality principles, the report describes how the PSP was developed, its principles, and its methods. Next is a summary of the PSP courses, the strategy used for teaching the PSP, selected data on PSP experience, PSP adoption in university curricula, and the status of PSP introduction into industry. The report concludes with comments on likely future trends involving the PSP.

[SM] Personal Software Process, PSP, Team Software Process, and TSP are service marks of Carnegie Mellon University.

1 Software Quality

Until shortly after World War II, the quality strategy in most industrial organizations was based almost entirely on testing. Groups typically established special quality departments to find and fix problems after products had been produced. It was not until the 1970s and 1980s that W. Edwards Deming and J.M. Juran convinced U.S. industry to focus on improving the way people did their jobs [Deming 82, Juran 88]. In the succeeding years, this focus on working processes has been responsible for major improvements in the quality of automobiles, electronics, or almost any other kind of product. The traditional test-and-fix strategy is now recognized as expensive, time-consuming, and ineffective for engineering and manufacturing work.

Even though most industrial organizations have now adopted modern quality principles, the software community has continued to rely on testing as the principal quality management method. For software, the first major step in the direction pioneered by Deming and Juran was taken by Michael Fagan when in 1976 he introduced software inspections [Fagan 76, Fagan 86]. By using inspections, organizations have substantially improved software quality. Another significant step in software quality improvement was taken with the initial introduction of the Capability Maturity Model® (CMM®) for software in 1987 [Humphrey 89, Paulk 95]. The CMM's principal focus was on the management system and the support and assistance provided to the development engineers. The CMM has had a substantial positive effect on the performance of software organizations [Herbsleb 97].

A further significant step in software quality improvement was taken with the Personal Software Process (PSP) [Humphrey 95]. The PSP extends the improvement process to the people who actually do the work—the practicing engineers. The PSP concentrates on the work practices of the individual engineers. The principle behind the PSP is that to produce quality software systems, every engineer who works on the system must do quality work.

The PSP is designed to help software professionals consistently use sound engineering practices. It shows them how to plan and track their work, use a defined and measured process, establish measurable goals, and track performance against these goals. The PSP shows engineers how to manage quality from the beginning of the job, how to analyze the results of each job, and how to use the results to improve the process for the next project.

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.

2 How the PSP Was Developed

After he led the initial development of the CMM for software, Watts Humphrey decided to apply CMM principles to writing small programs. Many people had been asking how to apply the CMM to small organizations or to the work of small software teams. While the CMM principles applied to such groups, more guidance was needed on precisely what to do. Humphrey decided to personally use CMM principles to develop module-sized programs both to see if the approach would work and to figure out how to convince software engineers to adopt such practices.

In developing module-sized programs, Humphrey personally used all of the software CMM practices up through Level 5. Shortly after he started this project in April 1989, the Software Engineering Institute (SEI) made Humphrey an SEI fellow, enabling him to spend full time on the PSP research. Over the next three years, he developed a total of 62 programs and defined about 15 PSP process versions. He used the Pascal, Object Pascal, and C++ programming languages to develop about 25,000 lines of code. From this experience, he concluded that the Deming and Juran process management principles were just as applicable to the work of individual software engineers as they were to other fields of technology.

Humphrey next wrote a textbook manuscript that he provided to several associates who planned to teach PSP courses. In September 1993, Howie Dow taught the first PSP course to four graduate students at the University of Massachusetts (Lowell). Humphrey also taught the PSP course during the winter semester of 1993-1994 at Carnegie Mellon University, as did Nazim Madhavji at McGill University and Soheil Khajanoori at Embry Riddle Aeronautical University. Based on the experiences and data from these initial courses, Humphrey revised the PSP textbook manuscript and published the final version in late 1994 [Humphrey 95]. At about the same time, Jim Over and Neil Reizer of the SEI and Robert Powels of Advanced Information Services (AIS) developed the first course to train instructors to teach the PSP in industry. Watts Humphrey and the SEI have continued working on PSP development and introduction by applying the same principles to the work of engineering teams. This work is called the Team Software Process.

3 The Principles of the PSP

The PSP design is based on the following planning and quality principles:

- Every engineer is different; to be most effective, engineers must plan their work and they must base their plans on their own personal data.
- To consistently improve their performance, engineers must personally use well-defined and measured processes.
- To produce quality products, engineers must feel personally responsible for the quality of their products. Superior products are not produced by mistake; engineers must strive to do quality work.
- It costs less to find and fix defects earlier in a process than later.
- It is more efficient to prevent defects than to find and fix them.
- The right way is always the fastest and cheapest way to do a job.

To do a software engineering job in the right way, engineers must plan their work before committing to or starting on a job, and they must use a defined process to plan the work. To understand their personal performance, they must measure the time that they spend on each job step, the defects that they inject and remove, and the sizes of the products they produce. To consistently produce quality products, engineers must plan, measure, and track product quality, and they must focus on quality from the beginning of a job. Finally, they must analyze the results of each job and use these findings to improve their personal processes.

4 The PSP Process Structure

The structure of the PSP process is shown conceptually in Figure 1. Starting with a requirements statement, the first step in the PSP process is planning. There is a planning script that guides this work and a plan summary for recording the planning data. While the engineers are following the script to do the work, they record their time and defect data on the time and defect logs. At the end of the job, during the postmortem phase (PM), they summarize the time and defect data from the logs, measure the program size, and enter these data in the plan summary form. When done, they deliver the finished product along with the completed plan summary form. A copy of the PSP1 plan summary is shown in Table 1.

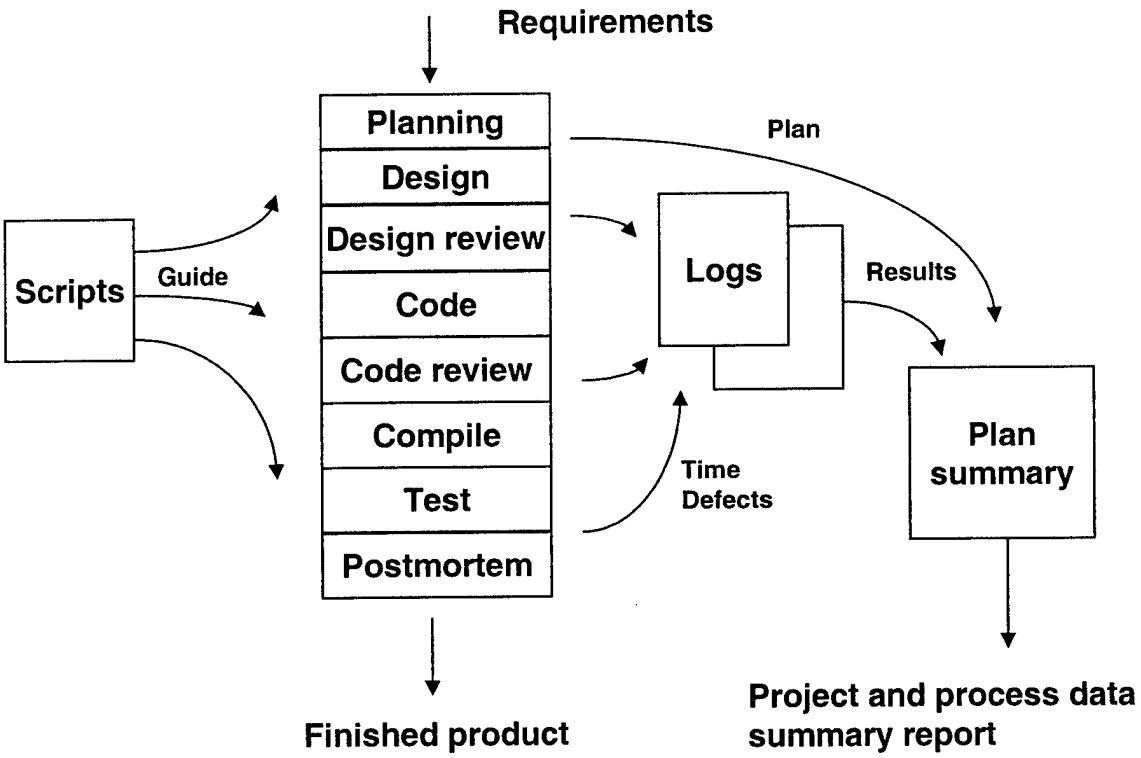


Figure 1: PSP Process Flow

Table 1: PSP1 Project Plan Summary

Student		Date	
Program		Program #	
Instructor		Language	

Summary LOC/Hour	***Plan***	***Actual***	***To Date***

Program Size (LOC):	***Plan***	**Actual**	**To Date**
Base(B)	(Measured)	(Measured)	
Deleted (D)	(Estimated)	(Counted)	
Modified (M)	(Estimated)	(Counted)	
Added (A)	(N – M)	(T – B + D – R)	
Reused (R)	(Estimated)	(Counted)	
Total New & Changed (N)	(Estimated)	(A + M)	
Total LOC (T)	(N + B – M – D + R)	(Measured)	
Total New Reused			
Total Object LOC (E)			

Time in Phase (min.)	**Plan**	**Actual**	**To Date**	**To Date %**
Planning				
Design				
Code				
Compile				
Test				
Postmortem				
Total				

Defects Injected	**Actual**	**To Date**	**To Date %**
Planning			
Design			
Code			
Compile			
Test			
Total Development			

Defects Removed	**Actual**	**To Date**	**To Date %**
Planning			
Design			
Code			
Compile			
Test			
Total Development			
After Development			

Since the PSP process has a number of methods that are not generally practiced by engineers, the PSP methods are introduced in a series of seven process versions. These versions are labeled PSP0 through PSP3, and each version has a similar set of logs, forms, scripts, and standards, as shown in Figure 2. The process scripts define the steps for each part of the process, the logs and forms provide templates for recording and storing data, and the standards guide the engineers as they do the work.

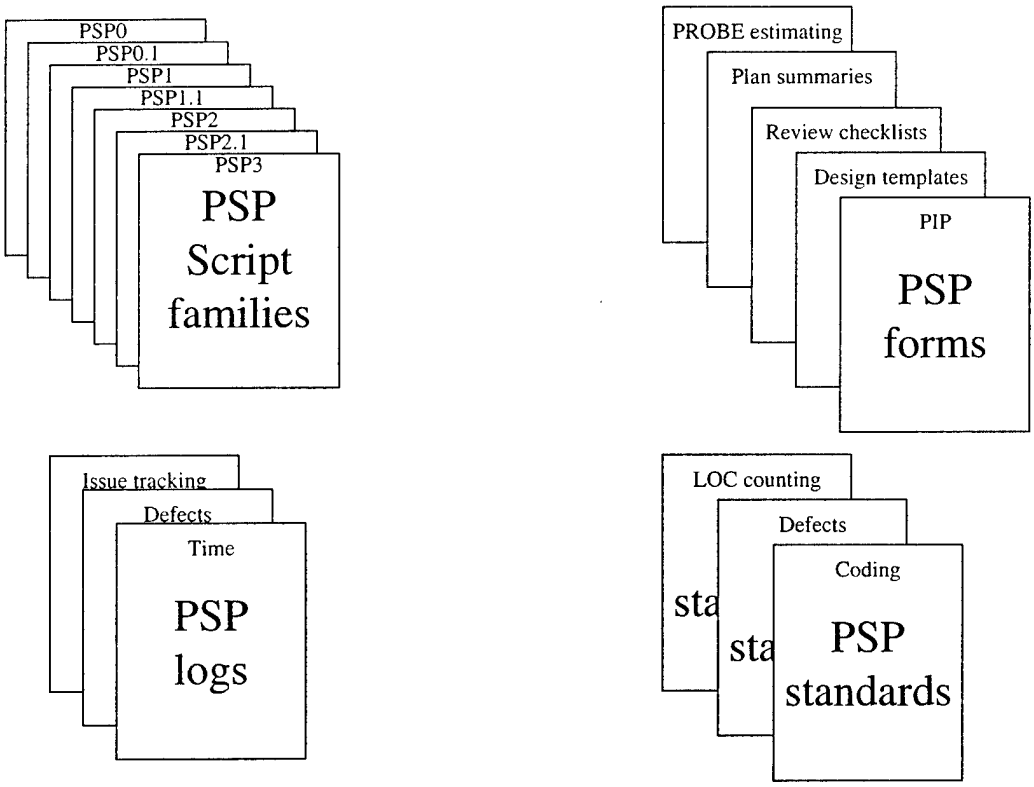


Figure 2: PSP Process Elements

An example PSP script is shown in Table 2. A PSP script is what Deming called an operational process [Deming 82]. In other words, it is a process that is designed to be used. It is constructed in a simple-to-use format with short and precise instructions. While scripts describe what to do, they are more like checklists than tutorials. They do not include the instructional materials that would be needed by untrained users. The purpose of the script is to guide engineers in consistently using a process that they understand. The next several sections of this report describe the various methods that the PSP uses for planning, estimating, data gathering, quality management, and design.

Table 2: PSP1 Process Script

Phase Number	Purpose	To guide you in developing module-level programs
	Entry Criteria	• Problem description • PSP1 Project Plan Summary form • ***Size Estimating Template*** • ***Historical estimate and actual size data*** • Time and Defect Recording Logs • Defect Type Standard • Stop watch (optional)
1	Planning	• Produce or obtain a requirements statement. • ***Use the PROBE method to*** estimate the total new and changed LOC required. • ***Complete the Size Estimate Template.*** • Estimate the required development time. • Enter the plan data in the Project Plan Summary form. • Complete the Time Recording Log.
2	Development	• Design the program. • Implement the design. • Compile the program and fix and log all defects found. • Test the program and fix and log all defects found. • Complete the Time Recording Log.
3	Postmortem	Complete the Project Plan Summary form with actual time, defect, and size data.
	Exit Criteria	• A thoroughly tested program • Completed Project Plan Summary form with estimated and actual data • ***Completed Size Estimating Template*** • ***Completed Test Report Template*** • Completed PIP forms • Completed Defect and Time Recording Logs

4.1 PSP Planning

The PSP planning process is shown in Figure 3. The following paragraphs describe the tasks in this figure.

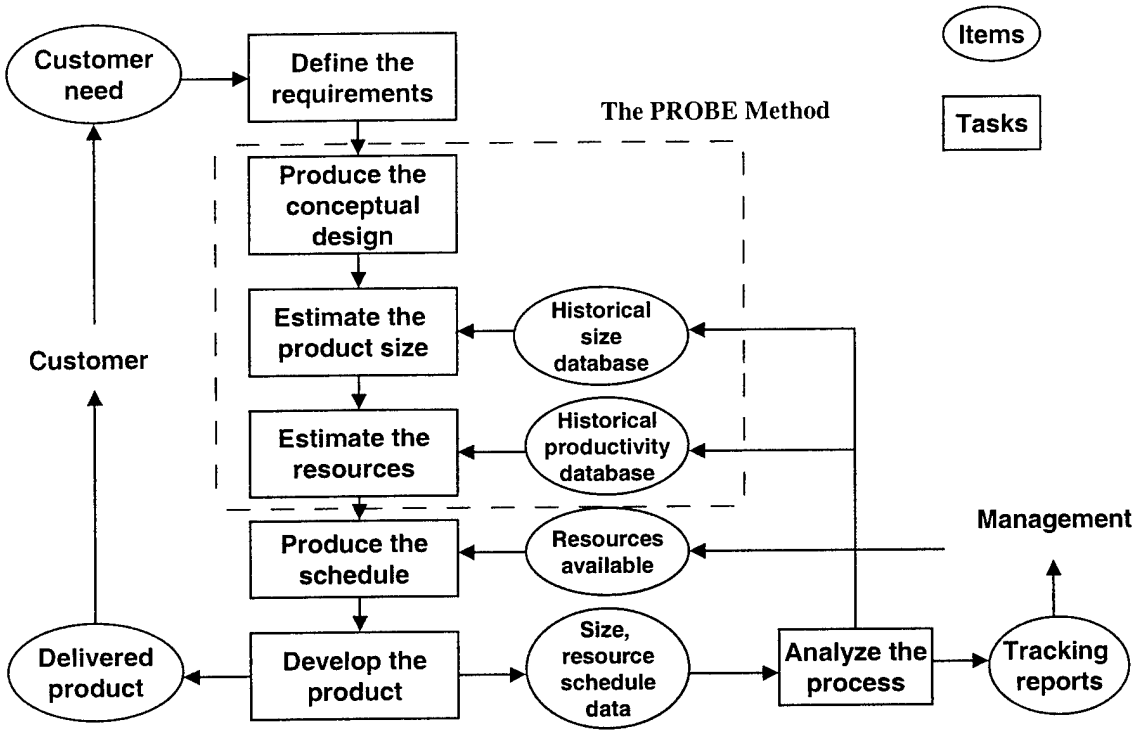


Figure 3: Project Planning Process

Requirements. Engineers start planning by defining the work that needs to be done in as much detail as possible. If all they have is a one-sentence requirements statement, then that statement must be the basis for the plan. Of course, the accuracy of the estimate and plan is heavily influenced by how much the engineers know about the work to be done.

Conceptual Design. To make an estimate and a plan, engineers first define how the product is to be designed and built. However, since the planning phase is too early to produce a complete product design, engineers produce what is called a conceptual design. This is a first, rough guess at what the product would look like if the engineers had to build it based on what they currently know. Later, during the design phase, the engineers examine design alternatives and produce a complete product design.

Estimate Product Size and Resources. The correlation of program size with development time is only moderately good for engineering teams and organizations. However, for individ-

ual engineers, the correlation is generally quite high. Therefore, the PSP starts with engineers estimating the sizes of the products they will personally develop. Then, based on their personal size and productivity data, the engineers estimate the time required to do the work. In the PSP, these size and resource estimates are made with the PROBE method.

4.2 Size Estimating with PROBE

PROBE stands for *PRO*xy *B*ased *E*stimating and it uses proxies or objects as the basis for estimating the likely size of a product [Humphrey 95]. With PROBE, engineers first determine the objects required to build the product described by the conceptual design. Then they determine the likely type and number of methods for each object. They refer to historical data on the sizes of similar objects they have previously developed and use linear regression to determine the likely overall size of the finished product. The example object size data in Table 3 show the five size ranges the PSP uses for objects. Since object size is a function of programming style, the PROBE method shows engineers how to use the data on the programs they have personally developed to generate size ranges for their personal use. Once they have estimated the sizes of the objects, they used linear regression to estimate the total amount of code they plan to develop. To use linear regression, the engineers must have historical data on estimated versus actual program size for at least three prior programs.

4.3 Calculation

Table 3: C++ Object Size Data[1]

C++ Object Sizes in LOC per Method					
Category	**Very Small**	**Small**	**Medium**	**Large**	**Very Large**
Calculation	2.34	5.13	11.25	24.66	54.04
Data	2.60	4.79	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
Logic	7.55	10.98	15.98	23.25	33.83
Set-up	3.88	5.04	6.56	8.53	11.09
Text	3.75	8.00	17.07	36.41	77.66

4.4 Resource Estimating with PROBE

The PROBE method also uses linear regression to estimate development resources. Again, this estimate is based on estimated size versus actual effort data from at least three prior projects. The data must demonstrate a reasonable correlation between program size and development time. The PSP requires that the r^2 for the correlation be at least 0.5.

[1] Reprinted from *A Discipline for Software Engineering*, page 117, by Watts S. Humphrey, Addison Wesley Publishing Co., 1995, Permission Required.

Once they have estimated the total time for the job, engineers use their historical data to estimate the time needed for each phase of the job. This is where the column *ToDate%* in the project plan summary in Table 1 is used. *ToDate%* keeps a running tally of the percentage of total development time that an engineer has spent in each development phase. Using these percentages as a guide, engineers allocate their estimated total development time to the planning, design, design review, code, code review, compile, unit test, and postmortem phases. When done, they have an estimate for the size of the program, the total development time, and the time required for each development phase.

Produce the Schedule. Once engineers know the time required for each process step, they estimate the time they will spend on the job each day or week. With that information, they spread the task time over the available scheduled hours to produce the planned time for completing each task. For larger projects, the PSP also introduces the earned-value method for scheduling and tracking the work [Boehm 81, Humphrey 95].

Develop the Product. In the step called *Develop the Product,* the engineers do the actual programming work. While this work is not normally considered part of the planning process, the engineers must use the data from this process to make future plans.

Analyze the Process. After completing a job, the engineers do a postmortem analysis of the work. In the postmortem, they update the project plan summary with actual data, calculate any required quality or other performance data, and review how well they performed against the plan. As a final planning step, the engineers update their historical size and productivity databases. At this time, they also examine any process improvement proposals (PIPs) and make process adjustments. They also review the defects found in compiling and testing, and update their personal review checklists to help them find and fix similar defects in the future.

5 PSP Data Gathering

In the PSP, engineers use data to monitor their work and to help them make better plans. To do this, they gather data on the time that they spend in each process phase, the sizes of the products they produce, and the quality of these products. These topics are discussed in the following sections.

5.1 Time Measures

In the PSP, engineers use the time recording log to measure the time spent in each process phase. In this log, they note the time they started working on a task, the time when they stopped the task, and any interruption time. For example, an interruption would be a phone call, a brief break, or someone interrupting to ask a question. By tracking time precisely, engineers track the effort actually spent on the project tasks. Since interruption time is essentially random, ignoring these times would add a large random error into the time data and reduce estimating accuracy.

5.2 Size Measures

Since the time it takes to develop a product is largely determined by the size of that product, when using the PSP, engineers first estimate the sizes of the products they plan to develop. Then, when they are done, they measure the sizes of the products they produced. This provides the engineers with the size data they need to make accurate size estimates. However, for these data to be useful, the size measure must correlate with the development time for the product. While lines of code (LOC) is the principal PSP size measure, any size measure can be used that provides a reasonable correlation between development time and product size. It should also permit automated measurement of actual product size.

5.2.1 Lines of Code (LOC)

The PSP uses the term "logical LOC" to refer to a logical construct of the programming language being used. Since there are many ways to define logical LOC, engineers must precisely define how they intend to measure LOC [Park 92]. When engineers work on a team or in a larger software organization, they should use the team's or organization's LOC standard. If there is no such standard, the PSP guides the engineers in defining their own. Since the PSP requires that engineers measure the sizes of the programs they produce, and since manually counting program size is both time consuming and inaccurate, the PSP also guides engineers in writing two automated LOC counters for use with the PSP course.

5.2.2 Size Categories

To track how the size of a program is changed during development, it is important to consider various categories of product LOC. These categories are

- **Base.** When an existing product is enhanced, base LOC is the size of the original product version before any modifications are made.
- **Added.** The added code is that code written for a new program or added to an existing base program.
- **Modified**. The modified LOC is that base code in an existing program that is changed.
- **Deleted**. The deleted LOC is that base code in an existing program that is deleted.
- **New and Changed.** When engineers develop software, it takes them much more time to add or modify a LOC than it does to delete or reuse one. Thus, in the PSP, engineers use only the added or modified code to make size and resource estimates. This code is called the New and Changed LOC.
- **Reused.** In the PSP, the reused LOC is the code that is taken from a reuse library and used, without modification, in a new program or program version. Reuse does not count the unmodified base code retained from a prior program version and it does not count any code that is reused with modifications.
- **New reuse**. The new reuse measure counts the LOC that an engineer develops and contributes to the reuse library.
- **Total**. The total LOC is the total size of a program, regardless of the source of the code.

5.2.3 Size Accounting

When modifying programs, it is often necessary to track the changes made to the original program. These data are used to determine the volume of product developed, the engineer's productivity, and product quality. To provide these data, the PSP uses the size accounting method to track all the additions, deletions, and changes made to a program [Humphrey 95].

To use size accounting, engineers need data on the amount of code in each size category. For example, if a product of 100,000 LOC were used to develop a new version, and there were 12,000 LOC of deleted code, 23,000 LOC of added code, 5,000 LOC of modified code, and 3,000 LOC of reused code, the New and changed LOC would be

N&C LOC = Added + Modified

28,000 = 23,000 + 5,000

When measuring the total size of a product, the calculations are as follows:

Total LOC = Base – Deleted + Added + Reused

Neither the modified nor the "new reuse" LOC are included in the total. This is because a modified LOC can be represented by a deleted and an added LOC, and the "new reuse" LOC are already accounted for in the added LOC. Using this formula, the total LOC for the above example would be

$$\text{Total} = 100{,}000 - 12{,}000 + 23{,}000 + 3{,}000 = 114{,}000 \text{ LOC}$$

This is 114 KLOC, where KLOC stands for 1,000 LOC.

5.3 Quality Measures

The principal quality focus of the PSP is on defects. To manage defects, engineers need data on the defects they inject, the phases in which they injected them, the phases in which they found and fixed them, and how long it took to fix them. With the PSP, engineers record data on every defect found in every phase, including reviews, inspections, compiling, and testing. These data are recorded in the defect recording log.

The term *defect* refers to something that is wrong with a program. It could be a misspelling, a punctuation mistake, or an incorrect program statement. Defects can be in programs, in designs, or even in the requirements, specifications, or other documentation. Defects can be found in any process phase, and they can be redundant or extra statements, incorrect statements, or omitted program sections. A defect, in fact, is anything that detracts from the program's ability to completely and effectively meet the user's needs. Thus a defect is an objective thing. It is something that engineers can identify, describe, and count.

With size, time, and defect data, there are many ways to measure, evaluate, and manage the quality of a program. The PSP provides a set of quality measures that helps engineers examine the quality of their programs from several perspectives. While no single measure can adequately indicate the overall quality of a program, the aggregate picture provided by the full set of PSP measures is a generally reliable quality indicator. The principal PSP quality measures are

- Defect density
- Review rate
- Development time ratios
- Defect ratios
- Yield
- Defects per hour
- Defect removal leverage
- Appraisal to failure ratio (A/FR)

Each of these measures is described in the following paragraphs.

Defect Density. Defect density refers to the defects per new and changed KLOC found in a program. Thus, if a 150 LOC program had 18 defects, The defect density would be

1000*18/150 = 120 defects/KLOC

Defect density is measured for the entire development process and for specific process phases. Since testing only removes a fraction of the defects in a product, when there are more defects that enter a test phase, there will be more remaining after the test phase is completed. Therefore, the number of defects found in a test phase is a good indication of the number that remain in the product after that test phase is completed.

Figure 4 shows IBM data on a major product that demonstrates this relationship [Kaplan 94]. These data imply that when engineers find relatively fewer defects in unit test, assuming that they have performed a competent test, their programs are of relatively higher quality. In the PSP, a program with five or fewer defects/KLOC in unit test is considered to be of good quality. For engineers who have not been PSP trained, typical unit test defect levels range from 20 to 40 or more defects/KLOC.

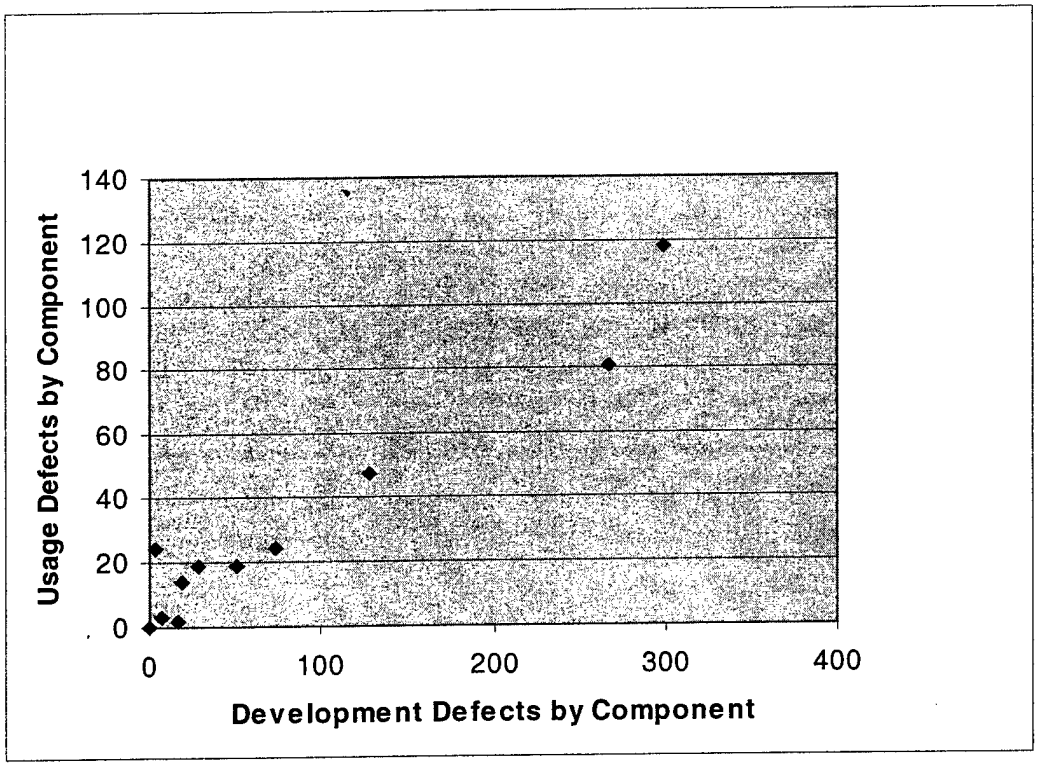


Figure 4: Development vs. Usage Defects; IBM Release 1 (r^2=.9267)

Review Rate. In the PSP design and code reviews, engineers personally review their programs. The PSP data show that when engineers review designs or code faster than about 150 to 200 new and changed LOC per hour, they miss many defects. With the PSP, engineers gather data on their reviews and determine how fast they should personally review programs to find all or most of the defects.

Development Time Ratios. Development time ratios refer to the ratio of the time spent by an engineer in any two development phases. In the PSP, the three development time ratios used in process evaluation are design time to coding time, design review time to design time, and code review time to coding time.

The design time to coding time measure is most useful in determining the quality of an engineer's design work. When engineers spend less time in designing than in coding, they are producing most of the design while coding. This means that the design is not documented, the design cannot be reviewed or inspected, and design quality is probably poor. The PSP guideline is that engineers should spend at least as much time producing a detailed design as they do in producing the code from that design.

The suggested ratio for design review time to design time is based on the PSP data shown in Table 4. In the PSP courses, engineers injected an average of 1.76 defects per hour in detailed design and found an average of 2.96 defects per hour in design reviews. Thus, to find all the defects that they injected during the design phase, engineers should spend 59% as much time in the design review as they did in the design. The PSP guideline is that engineers should spend at least half as much time reviewing a design as they did producing it.

Table 4: PSP Defect Injection and Removal Data

Phase	Injected/Hour	Removed/Hour
Design	1.76	0.10
Design Review	0.11	2.96
Coding	4.20	0.51
Code Review	0.11	6.52
Compile	0.60	15.84
Unit Test	0.38	2.21

Note: These data are from PSP courses where the exercises produced 2,386 programs with 308,023 LOC, 15,534 development hours, and 22,644 defects.

The code review time to coding time ratio is also a useful measure of process quality. During coding, engineers injected an average of 4.20 defects per hour and they found an average of 6.52 defects per hour in code reviews. Therefore, based on the PSP data in Table 4, engineers should spend about 65% as much time in code reviews as they do in coding. The PSP uses 50% as an approximate guideline for the code review time to coding time ratio.

Defect Ratios. The PSP defect ratios compare the defects found in one phase to those found in another. The principal defect ratios are defects found in code review divided by defects found in compile, and defects found in design review divided by defects found in unit test. A reasonable rule of thumb is that engineers should find at least twice as many defects when reviewing the code as they find in compiling it. The number of defects found while compiling is an objective measure of code quality. When engineers find more than twice as many defects in the code review as in compiling, it generally means that they have done a competent code review or that they did not record all the compile defects. The PSP data also suggest that the design review to unit test defect ratio be two or greater. If engineers find twice as many defects during design review as in unit test, they have probably done acceptable design reviews.

The PSP uses these two measures in combination with time ratios and defect density measures to indicate whether or not a program is ready for integration and system testing. The code review to compile defect ratio indicates code quality, and the design review to unit test defect ratio indicates design quality. If either measure is poor, the PSP quality criteria suggest that the program is likely to have problems in test or later use.

Yield. In the PSP, yield is measured in two ways. Phase yield measures the percentage of the total defects that are found and removed in a phase. For example, if a program entered unit test with 20 defects and unit testing found 9, the unit test phase yield would be 45%. Similarly, if a program entered code review with 50 defects and the review found 28, the code review phase yield would be 56%. Process yield refers to the percentage of the defects removed before the first compile and unit test. Since the PSP objective is to produce high quality programs, the suggested guideline for process yield is 70% better.

The yield measure cannot be precisely calculated until the end of a program's useful life. By then, presumably, all the defects would have been found and reported. When programs are of high quality, however, reasonably good early yield estimates can usually be made. For example, if a program had the defect data shown in Table 5, the PSP guideline for estimating the yield is to assume that the number of defects remaining in the program is equal to the number found in the last testing phase. Thus, in Table 5, it is likely that seven defects remain after unit testing.

Table 5: Example Defects Removed

Phase	Defects Removed
Design Review	11
Code Review	28
Compile	12
Unit Test	7
Total	58

To calculate yield, engineers use the data on the phases in which the defects were injected. They also need to assume that the defects remaining after unit test were injected in the same phases as those found during unit test. The data in Table 6 include one such defect in coding and six in detailed design.

Table 6: Example Defects Injected and Removed

Phase	Defects Injected	Defects Removed
Detailed Design	26	0
Design Review	0	11
Code	39	0
Code Review	0	28
Compile	0	12
Unit Test	0	7
After Unit Test	0	7
Total	65	65

As shown in Table 7, the total defects injected, including those estimated to remain, is 65. At design review entry, the defects present were 26, so the yield of the design review was 100*11/26 = 42.3%. At code review entry, the defects present were the total of 65 less those removed in design review, or 65 – 11 = 54, so the code review yield was 100*28/54 = 51.9%. Similarly, the compile yield was 100*12/(65-11-28) = 46.2%. Process yield is the best overall measure of process quality. It is the percentage of defects injected before compile that are removed before compile. For the data in Table 7, the process yield is 100*39/65 = 60.0%.

Table 7: Yield Calculations

Phase	Defects Injected	Defects Removed	Defects at Phase Entry	Phase Yield
Detailed Design	26	0	0	
Design Review	0	11	26	42.3%
Code	39	0	15	
Code Review	0	28	54	51.9%
Compile	0	12	26	46.2%
Unit Test	0	7	14	50.0%
After Unit Test	0	7	7	
Total	65	65		

Defects per Hour. With the PSP data, engineers can calculate the numbers of defects they inject and remove per hour. They can then use this measure to guide their personal planning. For example, if an engineer injected four defects per hour in coding and fixed eight defects per hour in code reviews, he or she would need to spend about 30 minutes in code reviews for every hour spent in coding. It would take this long to find and fix the defects that were

most likely injected. The defects per hour rates can be calculated from the data in the project plan summary form.

Defect Removal Leverage (DRL). Defect removal leverage measures the relative effectiveness of two defect removal phases. For instance, from the previous examples, the defect removal leverage for design reviews over unit test is 3.06/1.71 = 1.79. This means that the engineer will be 1.79 times more effective at finding defects in design reviews as in unit testing. The DRL measure helps engineers design the most effective defect removal plan.

A/FR. The appraisal to failure ratio (A/FR) measures the quality of the engineering process, using cost-of-quality parameters [Juran 88]. The *A* stands for the appraisal quality cost, or the percentage of development time spent in quality appraisal activities. In PSP, the appraisal cost is the time spent in design and code reviews, including the time spent repairing the defects found in those reviews.

The *F* in A/FR stands for the failure quality cost, which is the time spent in failure recovery and repair. The failure cost is the time spent in compile and unit test, including the time spent finding, fixing, recompiling, and retesting the defects found in compiling and testing.

The A/FR measure provides a useful way to assess quality, both for individual programs and to compare the quality of the development processes used for several programs. It also indicates the degree to which the engineer attempted to find and fix defects early in the development process. In the PSP course, engineers are told to plan for A/FR values of 2.0 or higher. This ensures that they plan adequate time for design and code reviews.

6 PSP Quality Management

Software quality is becoming increasingly important and defective software is an increasing problem [Leveson 95]. Any defect in a small part of a large program could potentially cause serious problems. As systems become faster, increasingly complex, and more automatic, catastrophic failures are increasingly likely and potentially more damaging [Perrow 84]. The problem is that the quality of large programs depends on the quality of the smaller parts of which they are built. Thus, to produce high-quality large programs, every software engineer who develops one or more of the system's parts must do high-quality work. This means that all engineers must manage the quality of their personal work. To help them do this, the PSP guides engineers in tracking and managing every defect.

6.1 Defects and Quality

Quality software products must meet the users' functional needs and perform reliably and consistently. While software functionality is most important to the program's users, the functionality is not usable unless the software runs. To get the software to run, however, engineers must remove almost all of its defects. Thus, while there are many aspects to software quality, the engineer's first quality concern must necessarily be on finding and fixing defects. PSP data show that even experienced programmers inject a defect in every seven to ten lines of code [Hayes 97]. Although engineers typically find and fix most of these defects when compiling and unit testing programs, traditional software methods leave many defects in the finished product.

Simple coding mistakes can produce very destructive or hard-to-find defects. Conversely, many sophisticated design defects are often easy to find. The mistake and its consequences are largely independent. Even trivial implementation errors can cause serious system problems. This is particularly important since the source of most software defects is simple programmer oversights and mistakes. While design issues are always important, newly developed programs typically have few design defects compared to the large number of simple mistakes. To improve program quality, PSP training shows engineers how to track and manage all of the defects they find in their programs.

6.2 The Engineer's Responsibility

The first PSP quality principle is that engineers are personally responsible for the quality of the programs they produce. Because the software engineer who writes a program is most familiar with it, that engineer can most efficiently and effectively find, fix, and prevent its defects. The PSP provides a series of practices and measures to help engineers assess the quality

of the programs they produce and to guide them in finding and fixing all program defects as quickly as possible. In addition to quality measurement and tracking, the PSP quality management methods are early defect removal and defect prevention.

6.3 Early Defect Removal

The principal PSP quality objective is to find and fix defects before the first compile or unit test. The PSP process includes design and code review steps in which engineers personally review their work products before they are inspected, compiled, or tested. The principle behind the PSP review process is that people tend to make the same mistakes repeatedly. Therefore, by analyzing data on the defects they have made and constructing a checklist of the actions needed to find those mistakes, engineers can find and fix defects most efficiently.

PSP-trained engineers find an average of 6.52 defects per hour in personal code reviews and 2.96 defects per hour in personal design reviews. This compares with 2.21 defects found per hour in unit testing [Humphrey 98]. By using PSP data, engineers can both save time and improve product quality. For example, from average PSP data, the time to remove 100 defects in unit testing is 45 hours while the time to find that number of defects in code reviews is only 15 hours.

6.4 Defect Prevention

The most effective way to manage defects is to prevent their initial introduction. In the PSP, there are three different but mutually supportive ways to prevent defects. The first is to have engineers record data on each defect they find and fix. Then they review these data to determine what caused the defects and to make process changes to eliminate these causes. By measuring their defects, engineers are more aware of their mistakes, they are more sensitive to their consequences, and they have the data needed to avoid making the same mistakes in the future. The rapid initial decline in total defects during the first few PSP course programs indicates the effectiveness of this prevention method.

The second prevention approach is to use an effective design method and notation to produce complete designs. To completely record a design, engineers must thoroughly understand it. This not only produces better designs; it results in fewer design mistakes.

The third defect prevention method is a direct consequence of the second: with a more thorough design, coding time is reduced, thus reducing defect injection. PSP data for 298 experienced engineers show the potential quality impact of a good design. These data show that during design, engineers inject an average of 1.76 defects per hour, while during coding they inject 4.20 defects per hour. Since it takes less time to code a completely documented design, by producing a thorough design, engineers will correspondingly reduce their coding time. So, by producing thorough designs, engineers actually inject fewer coding defects [Humphrey 98].

7 PSP Design

While the PSP can be used with any design method, it requires that the design be complete. The PSP considers a design to be complete when it defines all four of the dimensions shown in the object specification structure shown in Table 8 [De Champeaux 93]. The way that these four templates correspond to the object specification structure is shown in Table 9. The PSP has four design templates that address these dimensions.

Table 8: Object Specification Structure

Object Specification	**Internal**	**External**
Static	Attributes Constraints	Inheritance Class Structure
Dynamic	State Machine	Services Messages

Table 9: PSP Template Structure

Object Specification Templates	**Internal**	**External**
Static	Logic Specification Template	Function Specification Template (Inheritance Class Structure)
Dynamic	State Machine Template	Functional Specification Template (User Interaction) Operational Scenario Template

- The *operational scenario template* defines how users interact with the system.
- The *function specification template* specifies the call-return behavior of the program's objects and classes as well as the inheritance class structure.
- The *state specification template* defines the program's state machine behavior.
- The *logic specification template* specifies the program's internal logic, normally in a pseudocode language.

For a complete description of these templates and how to use them, consult the reference [Humphrey 95].

8 PSP Discipline

A principal problem in any engineering field is getting engineers to consistently use the methods they have been taught. In the PSP, these methods are: following a defined process, planning the work, gathering data, and using these data to analyze and improve the process. While this sounds simple in concept, it is not easy in practice. This is why a principal focus of PSP introduction is on providing a working environment that supports PSP practices. This is the main reason that the SEI developed the Team Software Process: to provide the disciplined environment that engineers need to consistently use the PSP methods in practice.

9 Introducing the PSP

PSP introduction in universities starts with a PSP course in which students write 10 programs and complete 5 data analysis reports [Humphrey 95]. The university course typically takes a full semester, during which the students follow the PSP process shown in Figure 5 to complete 10 exercise programs. Students start with the PSP0 process, where they use their current programming practices, record the time they spend in each process phase, and log all the defects they find. The PSP process is enhanced through seven process versions, with students writing one or two programs with each PSP version. For each program, they use the process methods just introduced, as well as all of the methods introduced with the previous process versions. By the end of the course, the students have written 10 programs and have learned to use all of the PSP process methods. This course is designed for graduate and advanced undergraduate students.

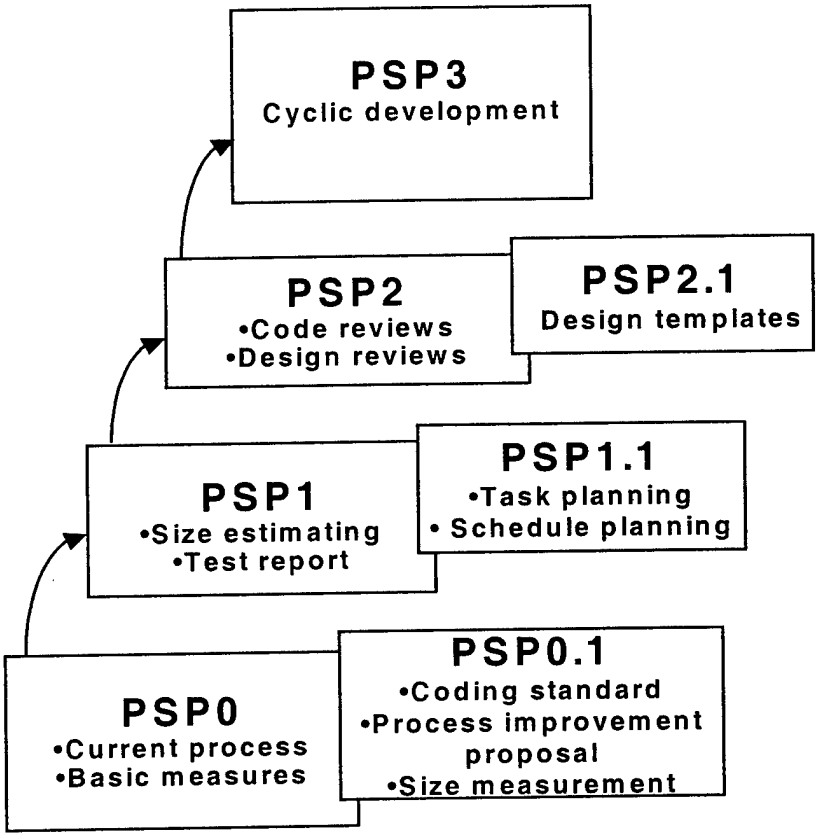


Figure 5: PSP/TSP Course Structure

There is also a PSP course for beginning university students [Humphrey 97]. While this introductory course teaches the same basic PSP principles as the advanced course, it is not as rigorous and students typically do not learn how to practice the full set of PSP methods. However, they do start using sound engineering practices with their first programming assignments. Then they are more likely to develop sound personal practices, and it is easier to teach them rigorous engineering practices when they later take the full PSP course.

When introducing the PSP in industry, engineers complete the full PSP course and write all 10 of the A-series of programming exercises in the PSP textbook [Humphrey 95]. This course takes about 120 to 150 engineering hours, and it is generally spread over 14 working days. After engineers are PSP trained, experience has shown that they must be properly managed and supported to consistently use the PSP in their work [Ferguson 97]. To help working engineers, teams, and managers consistently use the PSP methods, the SEI has developed the Team Software Process (TSP).

10 PSP Results

Hayes and Over have shown that the PSP course substantially improves engineering performance in estimating accuracy and early defect removal while not significantly affecting productivity [Hayes 97]. Typical results for estimating accuracy are shown in Figure 6. When engineers take the PSP course, they write 10 programs and gather data on their work. In Figure 6 and the following charts, performance on program 1 is shown at the left and the results achieved with program 10 are on the right. Since the students are using their prior programming practices with program 1 and the full set of PSP methods with program 10, the presumption is that the improvements result from the PSP methods. In writing these 10 exercise programs, defect levels also improved as shown in Figure 7. The productivity results are shown in Figure 8. All these data are averages for 298 students in SEI industrial courses on the PSP.

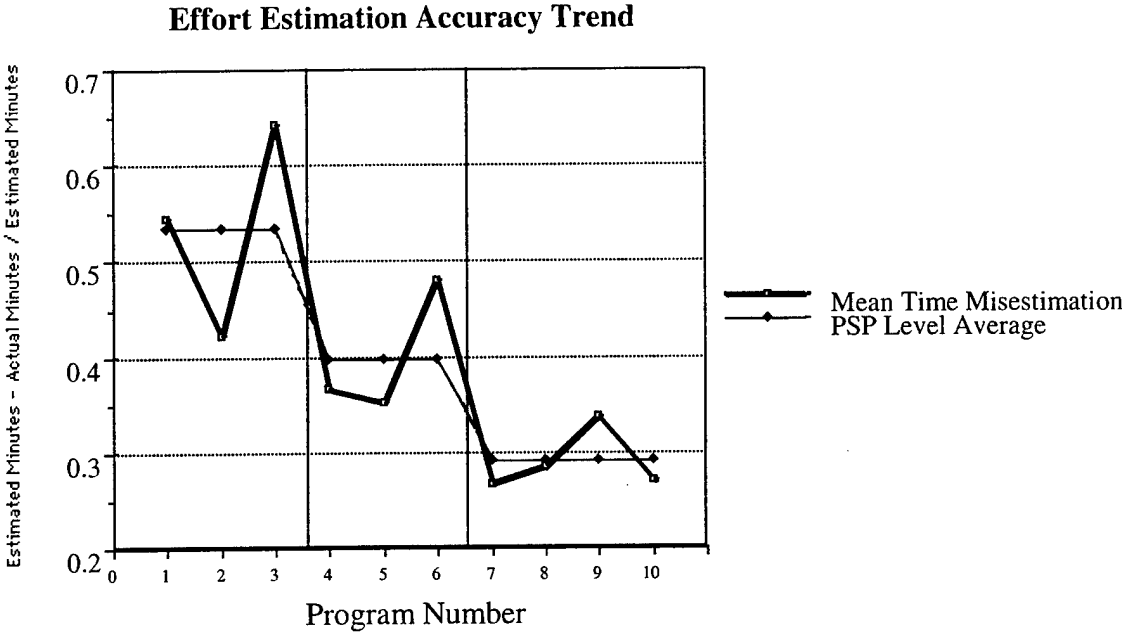


Figure 6: Effort Estimation Accuracy

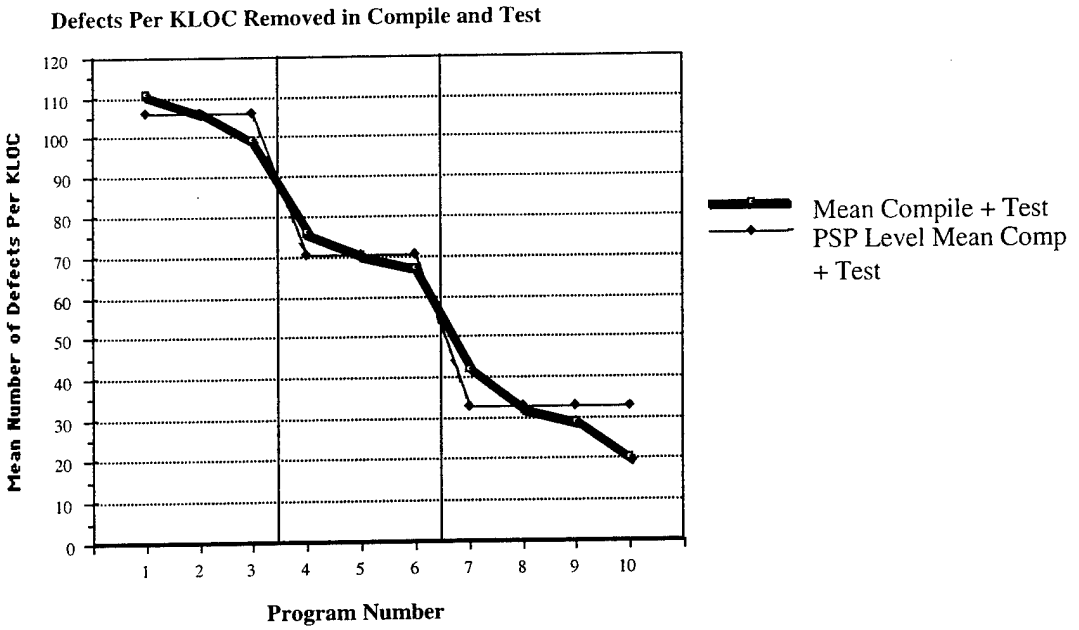


Figure 7: Defect Level Improvement

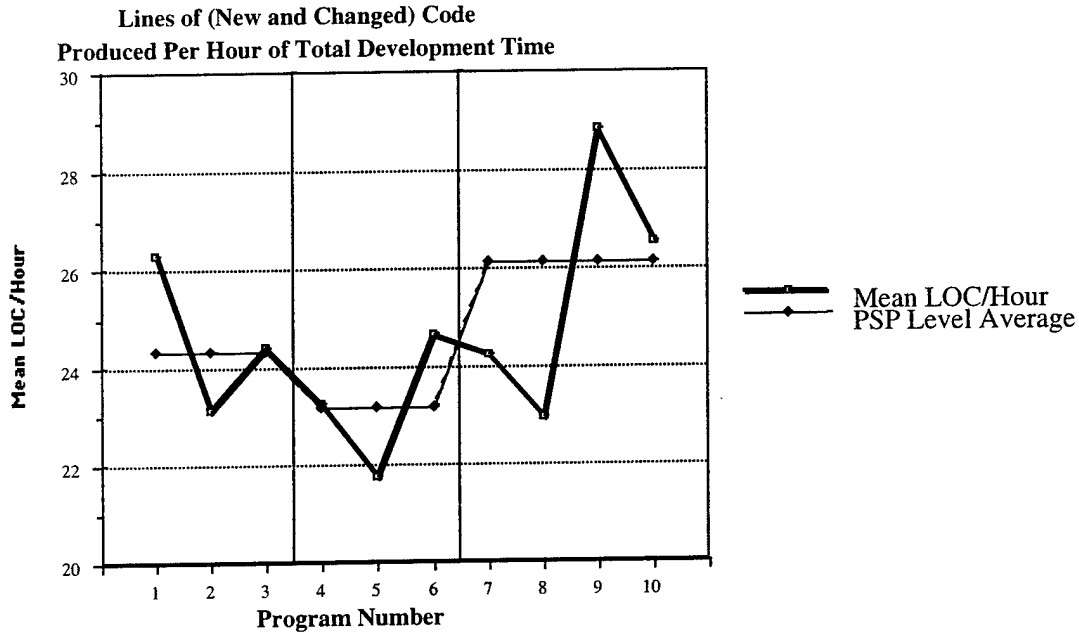


Figure 8: Productivity Results

While the PSP is relatively new, early industry results are becoming available [Ferguson 97, Seshagiri 00].

No controlled studies have yet compared the results obtained when students wrote identical programs, with an experimental group using the PSP methods and a control group not using the PSP. However, when taught by qualified instructors, student results consistently show significant improvement. Industrial data also show that engineering performance after PSP training is substantially better than before [Ferguson 97, Seshagiri 00]. It thus appears likely that PSP training was responsible for much of the improvement found in the PSP courses.

11 The PSP and Process Improvement

The PSP is one of a series of three complementary process-improvement approaches. These are

- The *Capability Maturity Model (CMM)*, which addresses software management issues [Humphrey 89, Paulk 95].
- The *PSP*, which is described in this report.
- The *Team Software Process (TSP)*, which guides PSP-trained engineers, teams, and managers in developing software-intensive products.

While these methods are all related, they address different aspects of organizational capability.

First, to produce superior products, the organization's management must have a sound business strategy and plan as well as a set of products that meet a market need. Without these, organizations cannot be successful, regardless of their other characteristics.

Second, to produce superior products, management must obtain superior performance from their engineers. This requires that they hire capable people and provide them with suitable leadership, processes, and support.

Third, the engineers must be able to work together effectively in a team environment and know how to consistently produce quality products.

Fourth, the engineering teams must be properly trained and capable of doing disciplined engineering work.

Without any one of these conditions, organizations are not likely to do superior work. In the above list, the CMM is designed to provide the second capability, the TSP the third, and the PSP the fourth. Unless engineers have the capabilities provided by PSP training, they cannot properly support their teams or consistently and reliably produce quality products. Unless teams are properly formed and led, they cannot manage and track their work or meet schedules. Finally, without proper management and executive leadership, organizations cannot succeed, regardless of their other capabilities.

12 PSP Status and Future Trends

In the future, software engineering groups will increasingly be required to deliver quality products, on time, and for their planned costs. Guaranteed reliability and service levels, warranted security, and contract performance penalties will be normal and engineering staffs that cannot meet such commitments will not survive.

While superior technical work will continue to be required, the performance of each individual engineer will be recognized as important. Quality systems require quality parts, and unless every engineer strives to produce quality work, the team cannot do so. Quality management will be an integral part of software engineering training. Engineers will have to learn how to measure the quality of their work and how to use these measures to produce essentially defect free work.

The PSP is designed to provide the disciplined practices software professionals will need in the future. While some industrial organizations are introducing these methods, broader introduction of disciplined methods must start in universities. Academic introduction of the PSP is currently supported with courses at both introductory and advanced levels [Humphrey 95, Humphrey 97]. Several universities in the U.S., Europe, and Australia now offer the PSP, and several institutions in Asia are considering its introduction. The SEI, in conjunction with various universities, also supports a one-week summer workshop for faculty who teach or wish to teach the PSP.

Industrial PSP introduction is also supported by the SEI with a PSP course for engineers and instructors. This course is offered several times a year in a condensed format tailored to industrial needs. The SEI also qualifies PSP instructors to assist organizations in introducing the PSP and it maintains a registry of qualified PSP instructors [SEI 00].

While the PSP is relatively new, the early results are promising. Both industrial use and academic adoption are increasing. Assuming that these trends continue, the future should see a closer integration of the PSP, TSP, and CMM methods and a closer coupling of the PSP academic courses with the broader computer science and software engineering curricula.

References

[Boehm 81] Boehm, B. *Software Engineering Economics.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[De Champeaux 93] De Champeaux, D., Lea, D., and Faure, P. *Object-Oriented System Development*, Reading MA: Addison-Wesley, 1993.

[Deming 82] Deming, W. E. *Out of the Crisis.* MIT Center for Advanced Engineering Study, Cambridge, MA, 1982.

[Fagan 76] Fagan, M. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal*, 15, 3 (1976).

[Fagan 86] Fagan, M. "Advances in Software Inspections." *IEEE Transactions on Software Engineering,* SE-12, 7, (July 1986).

[Ferguson 97] Ferguson, P., Humphrey, W., Khajenoori, S., Macke, S., and Matvya, A. "Introducing the Personal Software Process: Three Industry Case Studies," *IEEE Computer*, 30, 5 (May 1997): 24-31.

[Hayes 97] Hayes, W. and Over, J. *The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers* (CMU/SEI-97-TR-001), Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997 <http://www.sei.cmu.edu /pub/documents/97.reports/pdf/97tr001.pdf>.

[Herbsleb 97] Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W., and Paulk, M. "Software Quality and the Capability Maturity Model," *Communications of the ACM*, 40, 6 (June 1997): 30-40.

[Humphrey 89] Humphrey, W. *Managing the Software Process.* Reading, MA: Addison-Wesley, 1989.

[Humphrey 95] Humphrey, W. *A Discipline for Software Engineering.* Reading, MA: Addison-Wesley, 1995.

[Humphrey 97] Humphrey, W. *Introduction to the Personal Software Process.*

Reading MA: Addison-Wesley, 1997.

[Humphrey 98] Humphrey, W. "The Software Quality Index," *Software Quality Professional*, 1, 1 (December 1998):. 8-18.

[Juran 88] Juran, J. and Gryna, F. *Juran's Quality Control Handbook, Fourth Edition.* New York: McGraw-Hill Book Company, 1988.

[Kaplan 94] Kaplan, C., Clark, R., and Tang, V. *Secrets of Software Quality, 40 Innovations from IBM*. New York, N.Y.: McGraw-Hill, Inc., 1994.

[Leveson 95] Leveson, N. *Safeware, System Safety and Computers*. Reading, MA: Addison Wesley, 1995.

[Park 92] Park, R. *Software Size Measurement: A Framework for Counting Source Statements* (CMU/SEI-92-TR-20, ADA258304). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University (Sept. 1992) <http://www.sei.cmu.edu/pub/documents/92.reports /pdf/tr20.92.pdf>.

[Paulk 95] Paulk, M., et al. *The Capability Maturity Model: Guidelines for Improving the Software Process.* Reading, MA: Addison Wesley, 1995.

[Perrow 84] Perrow, C. *Normal Accidents, Living with High-Risk Technologies.* New York, NY: Basic Books, Inc., 1984.

[SEI 00] "Building High Performance Teams Using Team Software Process (TSP) and Personal Software Process (PSP)." Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University <http://www.sei.cmu.edu/tsp>.

[Seshagiri 00] Seshagiri, G. "Making Quality Happen: The Managers' Role, AIS Case Study," *Crosstalk* (June 2000).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	November 2000	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
The Personal Software Process℠ (PSP℠)	F19628-00-C-0003

6. AUTHOR(S)

Watts S. Humphrey

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2000-TR-022

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	ESC-TR-2000-022

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

The Personal Software Process℠ (PSP℠) provides engineers with a disciplined personal framework for doing software work. The PSP process consists of a set of methods, forms, and scripts that show software engineers how to plan, measure, and manage their work. It is introduced with a textbook and a course that are designed for both industrial and academic use. The PSP is designed for use with any programming language or design methodology and it can be used for most aspects of software work, including writing requirements, running tests, defining processes, and repairing defects. When engineers use the PSP, the recommended process goal is to produce zero-defect products on schedule and within planned costs. When used with the Team Software Process℠ (TSP℠), the PSP has been effective in helping engineers achieve these objectives.

This report describes in detail what the PSP is and how it works. Starting with a brief discussion of the relationship of the PSP to general quality principles, the report describes how the PSP was developed, its principles, and its methods. Next is a summary of the PSP courses, the strategy used for teaching the PSP, selected data on PSP experience, PSP adoption in university curricula, and the status of PSP introduction into industry. The report concludes with comments on likely future trends involving the PSP.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Personal Software Process, PSP, Team Software Process, TSP, software development, software engineering	54

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL